Approved for public release; distribution is unlimited

NPSNET: Real-Time Walkthrough and Rendering of Urban Terrain

by

Mark R. Boone
Civilian, DoD, NSWC Dahlgren, VA
B.S., University of Missouri-Rolla, 1987

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1993

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): CS	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNI ACCESSIO

11. TITLE (Include Security Classification)
NPSNET: Real-Time Walkthrough and Rendering of Urban Terrain

12. PERSONAL AUTHOR(S)
Mark R. Boone

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM 9/91 TO 12/92	June 1993	49

16. SUPPLEMENTARY NOTATION: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Virtual Worlds, Radiosity, Walkthrough

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The focus of this thesis is performing walkthroughs of urban environments in real-time. This encompasses environment both inside and outside the building, whereby a user can traverse the external virtual world and enter any building and move throughout the interior. While inside the building, the user can see the exterior world any windows present. The transition from one environment to the other is transparent to the user.

The 2D terrain file used as source data is converted and rendered in 3D. This 3D representation of the model then be used for Military Operations in Urban Terrain (MOUT) training. To help maintain the real-time respons new methods of visibility determination are used to lessen the overall polygon flow through the graphics pipeli The radiosity lighting model is used for the walkthrough of building interiors.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
[X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL: Michael J. Zyda; Dave R. Pratt	22b. TELEPHONE (Include Area Code): (408) 656-2305	22c. OFFICE SYMBOL: CS/Zk

ABSTRACT

The focus of this thesis is performing walkthroughs of urban environments in real-time. This encompasses the environment both inside and outside of the building, whereby a user can traverse the external virtual world and then enter any building and move throughout the interior. While inside the building, the user can see the exterior world via any windows present. The transition from one environment to the other is transparent to the user.

The 2D terrain file used as source data is converted and rendered in 3D. This 3D representation of the model can then be used for Military Operations in Urban Terrain (MOUT) training. To help maintain the real-time response, new methods of visibility determination are used to lessen the overall polygon flow through the graphics pipeline. The radiosity lighting model is used for the walkthrough of building interiors.

TABLE OF CONTENTS

I. INTRODUCTION

A. PREMISE

Many 3D graphical combat simulators exist today, such as SIMNET [Garv88] and NPSNET [Zyda92]. These simulators model combat at a high resolution level, where the basic units in the model are individual tanks, planes, ships, etc. and not groups of such (i.e. brigades, divisions, etc.). Specific actions for each of these entities, such as search, detection, and firing are individually modeled. This level of detail requires a very complex system not only for modeling, but also for graphically representing in 3D the entities involved. Moreover, increasing complexity is being built into these simulators as both graphics workstations and software packages improve their capabilities. These improvements help create a comprehensive environment which is as realistic and believable as possible with the current technology.

While most combat simulators have not reached true virtual reality (VR), advances in technology may make this feasible in the near future. One such area under investigation is the increase in the resolution of the combat model, so that a single soldier is the smallest entity represented. The user then takes the role of a soldier and moves through and interacts with the modelled virtual environment. This environment where the individual soldier plays a more significant role in the operation, such as combat in and around a city or town, is known as "urban terrain".

Within the urban terrain, combat can be modelled on the soldier-based level, modelling combat for both inside and outside the buildings. Conceivably, Military Operation in Urban Terrain (MOUT), such as hostage rescue, could be modeled and rehearsed using the actual floor plan of the building(s). Some aspects involved in such a simulation have already been investigated individually, for example building walkthroughs [Funk92][Aier90]. But such existing work has not been done on such a large scale as is required or with the intention of solving the problem outlined above.

Such a virtual world modeler as this has the advantage of modeling any urban terrain. Based on the scenario to be modeled, different urban data files can be used to represent the building or set of buildings. This simulator is more flexible and cost effective for MOUT training than the current method of physically building 'model'towns. Moreover, new physical model towns would not have to be built; instead just a different terrain data file could be used. Using different terrain files creates variations of a modeled town where small changes can create several towns rather than using manpower and expenses to physically create these models. Personnel need not travel to actual model towns; rather, they interact using graphical workstations locally and over a network.

B. REALIZATION

Such virtual world models are not built overnight but through a series of steps leading to the final model. One such step is the NPSNET Urban Combat Trainer (NUCAT), which provides a mechanism and framework for rendering and moving a single user through a complex virtual environment, namely an urban one. Thus, NUCAT presents the first few steps down this path by providing the basic foundation and proof of concept to build upon.

The NUCAT draws together several graphics areas of study and combines them into a cohesive whole, such as building walkthrough, radiosity, binary space partitioning (BSP) trees, and virtual worlds. Much focus is aimed at maintaining the real-time aspect along with allowing flexibility in using the various diverse areas. Thus, the NUCAT project is not definitively coupled with any specific area, rather it is loosely coupled so that competing areas may be both incorporated and investigated simultaneously. This also allows for a flexible use of varying levels of detail depending upon what data is available. For example, NUCAT is able to use existing radiosity files of a building, or, if the file does not exist, a polygon representation of the building is used.

Additional areas that were addressed are: construction and modelling of the virtual world and its contents, navigation, visibility determination, and image synthesis. There do not seem to be any projects that allow the user to move around in a virtual world, then to enter a building and move around inside of it, and to still see the outside world by looking out a window. The NUCAT project provides this capability in real-time.

The urban virtual environment is a highly complex one; for NUCAT it is simplified into two categories: exterior and interior. The exterior environment contains all objects outside of a building, objects such as buildings, vegetation, roads, terrain elevation, etc. This also refers to where the user is moving around outside of any buildings. The interior environment is the internal model for a building such as a floor plan, rooms, portals, furniture, etc., as well as movement of the user inside the building.

In addition to the complexity of the object representation for the urban terrain, the image quality and real-time constraints provide suitable realism. To move through the urban environment, a simple flying model is used. This flying model does not have any physically-based constraints tied to it.

Finally, the specific implementation of this initial framework is based on the Urban Combat Computer Assisted Training System (UCCATS) data files provided by the Lawrence Livermore National Laboratory, Livermore, CA [Dobb90]. The NUCAT project deals primarily with the visualization of the 2D UCCATS terrain data in 3D. NUCAT does not deal with or use any of the UCCATS combat models.

Other terrain data files of differing format and content conceivably can be used, since all data is pre-processed to generate a NUCAT-specific data file. Thus, to support such files, only a new pre-processor needs to be generated to handle the new file format and generate a NUCAT data file.

C. THESIS ORGANIZATION

Chapter I introduces the motivation for the NUCAT project and the approach used in solving the problem.

Chapter II briefly covers some of the prior work done in the areas of radiosity and building walkthroughs and provides background information for understanding the concepts used and referred to in the rest of the paper. Background information is given for radiosity, BSP trees, and the data files used by NUCAT.

Chapter III gives an overview of the NUCAT organization and program flow, and briefly discusses some of the more important concepts, such as virtual world construction, visibility determination, building walkthrough, and radiosity.

Chapter IV provides details on the various parts of the NUCAT project and related algorithms. In this chapter the following topics are discussed in detail: how the 3D objects are created and rendered, how the virtual world is navigated, two methods used for visibility determination, techniques used for creating realistic images, and how walkthroughs and radiosity are accomplished.

Chapter V contains concluding remarks on NUCAT, and outlines areas of future work to be done.

Appendix A contains the user manual for NUCAT. The user manual describes the various input devices and menu options available for interacting with NUCAT.

II. BACKGROUND

A. PREVIOUS WORK

NUCAT is a combination of several graphics topics which, in the past, have been studied individually and not combined into a whole. Two of these topics, radiosity and buildings walkthroughs, are discussed below.

1. Radiosity

Radiosity, a physically-based lighting model, is a topic of much study. Much investigation for determining methods of accelerating the radiosity calculations and for improving the accuracy of these calculations has been done. Several methods are used for accelerating the radiosity calculation. The most common method uses progressive refinement [Choe88], where the radiosity of a patch is shot into the environment rather than gathered. This allows the radiosity scene to be viewed immediately as the radiosity continues to be calculated rather than waiting for all the radiosity equations to be solved simultaneously before being able to render the scene. Another common method is using parallel processing [Baum90][Puec90], where each processor handles the calculation for one patch at a time. Thus, the radiosity for multiple patches is solved concurrently rather than having to wait for each one to be solved in sequence. The final method for increasing speed is using less precise resolution [Reck90], where a less precise method for calculating form factors is used. Therefore, some approximation method is used instead of solving integrals to determine exact radiosity values. The hemi-cube is most commonly used, although a hemi-sphere or hemi-plane can also be used. The number of cells the hemi-cube is divided into affects the precision and speed of calculation, where a greater number of hemi-cube cells allows greater precision yet is more time consuming.

While attempting to improve the accuracy of the radiosity calculations, solutions are often aimed at either fixing the aliasing errors resulting from use of the hemi-cube or providing alternate means of calculating the form factors. Several methods can be used to achieve this. One method is replacing the hemi-cube with a hemi-sphere, although more complicated to calculate, this way produces a more accurate form factor. Another method of offsetting some aliasing errors of the hemi-cube is subdividing the patches into elements. Patches that contain a sharp radiosity gradient are good candidates for being subdivided. The elements are only used for calculating the radiosity received, they are not used for shooting radiosity into the environment.

Other topics related to radiosity are preprocessing of the data model (in generating proper tmeshs) and adding specular highlights to the model. Tmesh data generated for radiosity use [Baum91] seeks to eliminate some of the anomalies that may otherwise occur, such as false shadow boundaries at T-vertices or shadow bleeding from underneath patches. For specular highlights, a two pass radiosity model is created; in pass one, the normal diffuse radiosity lighting is calculated; in pass two, the specular lighting is calculated and added to the scene [Fole90].

2. Walkthrough

The area of building walkthrough focuses on representing and modeling the interior of a building (using BSP trees or hierarchical) and determining what the user sees inside the building. How the building interior is modeled is strongly related to the method used for determining what is visible, which method is time consuming and best completed as a preprocessing step. This enables the rendering portion to be done in real-time.

Most previous work done in this area has been centered around one building as the environment. Sometimes, not even allowing the user outside of the building model or only allowing for a scripted movement inside.

Two predominate contributors to building walkthroughs are the University of California - Berkeley [Funk92][Tell91] and the University of North Carolina - Chapel Hill [Aire90][Broo86]. At UC-Berkeley, Teller and others have investigated using a hierarchical data structure for the building model and using pre-generated potentially visible sets (PVS) for determining what rooms are visible. Each room has its own PVS, which represents all possible rooms visible from any view point within a particular room.

UNC-Chapel Hill has implemented walkthroughs using BSP trees for modeling the building interior and also using PVSs for determining visibility. Using partitioning planes, the BSP tree is created by subdividing the building model into cells, where each node of the BSP tree contains one cell. These cells usually represent one room. For each cell, a PVS is created and stored with it in the BSP tree. This PVS represents all other cells that may possibly be seen from the current cell. UNC-Chapel Hill has also devoted much time to investigating various user input and viewing devices for use in the walkthroughs, such as head mounted displays (HMDs), treadmills, and large screen displays.

B. RADIOSITY BACKGROUND

Radiosity is a physical-based lighting model used for showing the diffuse illumination of an enclosed environment such as a building interior. Radiosity is defined as the rate at which energy, as light, leaves a

surface. For a particular scene, the radiosity lighting calculations only need to be done once; after which, the user can view the scene from any viewpoint without having to recalculate the lighting model. Since the lighting model is independent of the user viewpoint, radiosity is the preferred lighting model for building walkthroughs. The lighting is not recomputed whenever the user viewpoint changes as is necessary for ray tracing or for the IRIS GL lights, which have specular components for the lighting model. Since radiosity only models diffuse lighting, its calculations are not dependent on the user view point. Thus, not having to recalculate the lighting model for each frame, greatly improves the frame rate and enables the walkthrough to be viewed in real-time.

The scene to be lighted is represented as a set of patches. In the simplistic case, each polygon used in the scene is represented by one patch. Realistically, the polygons are subdivided in some manner to produce a set of quadrilateral or triangle patches for the radiosity package [Baum91]. Each patch emits and/or reflects diffuse light into the environment. Only patches representing light sources will emit radiosity, all other patches will act as reflectors.

The equation used to calculate the radiosity for one patch is:

$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_i F_{ji} \frac{A_j}{A_i}$$

where

B_i = radiosity of patch i (watts/m^2)
E_i = emission of patch i (watts/m^2)
p_i = reflectance of patch i
A_i = area of patch i (m^2)
A_j = area of patch j (m^2)
F_{ji} = form-factor from patch j to patch i
n = number of discrete patches in the environment

Thus, the radiosity for patch i is the sum of its own emission and the gathering of energy reflected from all the other patches in the scene.

From the above equation, the form factor F_{ji} specifies the amount of energy leaving patch j that lands on patch i. This can be expressed as a percentage from 0 to 1 of patch j's energy. Form factors are purely geometric and remain the same as long as the scene geometry remains the same. Calculation of form factors is often the most time consuming part of the radiosity calculation. The proper equation for calculating the form factor for patch i to patch j is:

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} \frac{d}{dA_j} \frac{d}{dA_i}$$

where,

r = length of a ray between the center points of both patches

θi = angle between ray r and patch Ai's normal vector

θj = angle between ray r and patch Aj's normal vector

A_i = area of patch i

A_j = area of patch j

Rather than calculating this double integral for each form factor, several approximation techniques which are faster and easier to implement can be used. The most common one in use is the hemi-cube. The hemi-cube is a half cube centered on the shooting patch and is subdivided into small square cells. To arrive at the form factor for a patch, the patch is projected onto the hemi-cube's five sides (Figure 2-1). Then the delta-form factors for all the square cells hit are summed to reach the overall form factor. Geometric forms other than a cube may be used, such as a hemi-sphere or a pyramid.

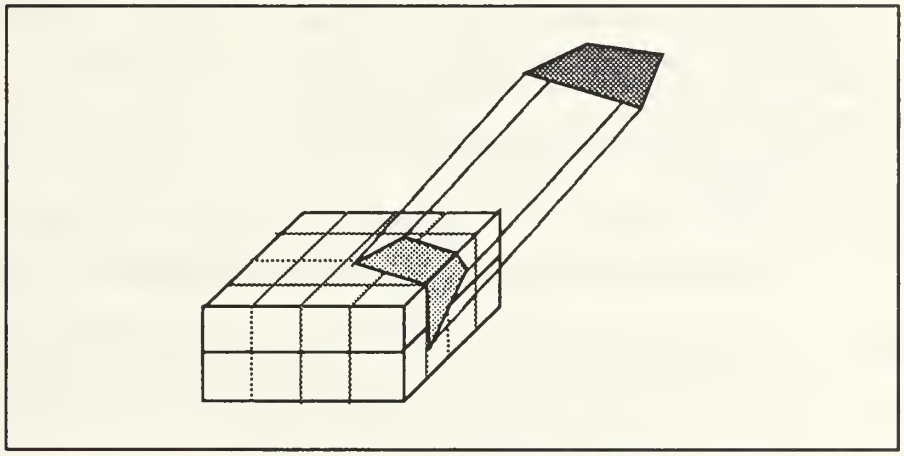

Figure 2-1. Polygon Projection Onto a Hemi-Cube

The equation for calculating the radiosity of a single patch is solved for every patch in the scene thus creating a set of simultaneous equations that must be solved. This can be quite daunting when the scene is made up of a 1000 patches or more. Therefore, another method, progressive refinement [Cohe88][Fole90], has been developed which avoids having to solve all the equations simultaneously.

The progressive refinement method shoots radiosity into the scene rather than gathering it as described above. Whereby, the patch with the most unshot radiosity is selected and its radiosity is shot (or distributed)

into the scene, thus updating the radiosity values of all patches upon which its radiosity falls. This continues until the shooting patch's unshot radiosity falls below some specified threshold, and the calculation stops. Figure 2-2 gives a basic outline of the progressive refinement algorithm.

An advantage of using progressive refinement is that the scene can be immediately viewed while the radiosity calculations continue. Thus, the scene will progressively become clearer as the calculations continue. In addition, an ambient term can be added to the scene to brighten the early repetitions. As more of the radiosity gets distributed throughout the scene, the ambient term gradually decreases to zero.

```
Select shooting patch i
Calculate form factor, F_ij, for each patch j
For each patch j
    Calculate amount of shooting radiosity that hits patch j (delta rad)
    Increment current patch rad by delta rad
    Increment unshot patch rad by delta rad
Set unshot rad for shooting patch i to zero
Display results
```

Figure 2-2. Progressive Refinement Algorithm

Patches can be subdivided into elements to provide better shading across the patch. The elements receive, rather than project, radiosity. For shooting radiosity, the original patch is used. Subdivision usually occurs around sharp shadow gradients, such as edges.

For a more in-depth discussion of radiosity please refer to any of the listed radiosity references. Also, see [Foley90] for a nice introduction to the topic.

C. BSP TREE BACKGROUND

A binary space partitioning (BSP) tree [Fole90][Fuch83] is a data structure used for representing a set of polygons. A BSP is a binary tree in which each node represents a polygon that partitions the remaining set of polygons into a front and a back half given by the child nodes. The plane in which the polygon lies partitions the space into a positive and negative half space. The positive half space is the front of the plane (i.e. direction of the planar normal) and the negative is the back.

Those polygons in the positive half space are grouped together under one child node, while those in the negative half space are contained in the other. Polygons that are split by the plane must be divided into two polygons, where one is the front and the other is the back.

A BSP tree is created by selecting one polygon from a set to be used as the partitioning plane. To select the best polygon for use requires some heuristics. Possible heuristics for selecting the best polygon to divide the remaining set are:

(1) Larger area polygons are better (less likely to get split later).

(2) Polygon selected evenly splits the set (for balanced trees).

(3) Polygon selected causes the least amount of other polygons to be split.

Often, all three heuristics are used in some weighted manner. After the partitioning polygon is selected, the remaining polygons are divided into a positive half space and a negative half space. These two sets are the child nodes for the selected polygon. The procedure is then recursively done for each child of the parent node. The final result is a BSP tree.

BSP trees are mainly used for visibility testing to minimize the number of polygons to be rendered. This is accomplished by taking the user's view point and direction and processing the BSP tree in the following manner: If the user is on the front side of the polygon, then the back side child if processed first; next, the root node; finally, the front side child. This order is reversed if the user is behind the polygon; front side child, root node, and then back side child. This tree traversal order provides a painter's like algorithm for drawing the polygons. Some child nodes may be culled from the traversal based on calculations for the parent node. Refer to Table 1 and corresponding Figure 2-3 below.

Table 1: BSP TREE CULLING

Case	View Side	Facing	Draw	Culling	Traversal
1	Front	Yes	Yes	None	B, R, F
2	Front	No	No	Back side	-,-,F
3	Back	Yes	Maybe	None	F,R,B
4	Back	No	No	Front side	-,-,B

Case 1: User is in front of the polygon and facing it: the back child is traversed, the polygon is drawn, and then the front child traversed.

Case 2: User is in front of the polygon, but not facing it: only the front child is traversed. The polygon is not drawn and the back child is ignored.

Case 3: User is behind the polygon and facing it: the front child is traversed, the polygon is drawn, and finally the back child is traversed. If backface removal is being used, then the polygon is not drawn.

Case 4: User is behind the polygon and not facing it: only the back side child is traversed. The polygon is not drawn and the front child is culled from the traversal.

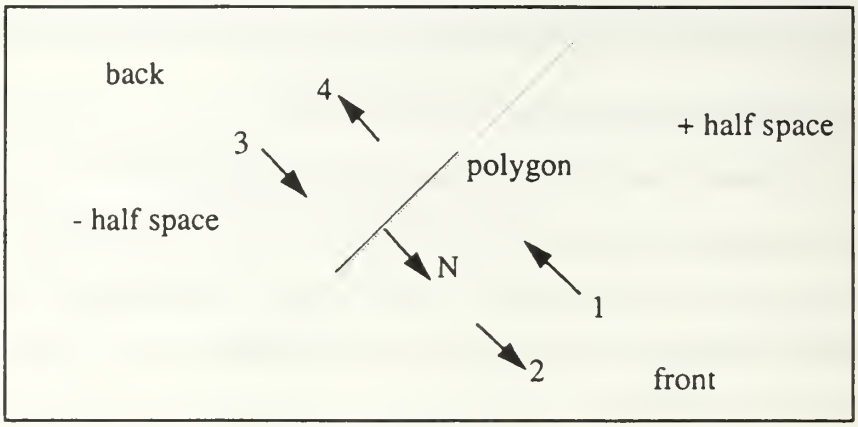

Figure 2-3. BSP Tree Culling

1. Project Note

A BSP tree was created and used in the project for comparison with an object-based method of drawing which polygons are visible when outside of a building. The BSP tree was created using the exterior wall polygons of all the buildings. When implemented, the BSP tree does not provide any increase in performance, yet it decreases by approximately 2 frames per second (fps). This is attributed to the large fragmentation of the source polygon set. Currently, BSP trees are not used in the model, but may again be investigated for use in modelling the interior of a building. This greatly affects the method used for visibility determination inside the building.

D. NUCAT DATA FILES

The NUCAT project uses the UCCATS data file as its main source of data [Dobb90]. This data file along with others used by NUCAT are described below. The specific UCCATS data file used for the project models a 1.5km x 1.5km piece of terrain which contains a town called Doughboy City which is located outside Berlin, Germany; this site is used by the U.S. Army for MOUT training.

1. UCCATS Data File

All the terrain data used by NUCAT originates from the UCCATS data files provided by the UCCATS project located at Lawrence Livermore National Laboratory, Livermore, CA. The data files provides all exterior data, which includes elevation, buildings, vegetation, roads, etc. It does not provide data for the interior of buildings, which is generated separately for each building. Note that this data file is 2D data from a top-down view which NUCAT interprets and renders in 3D.

There are four basic blocks of data in the UCCATS file which the NUCAT project uses.

(1) Map data. This data describes the map area being represented. It includes grid boundaries, size of each grid, elevation ranges, etc.

(2) Elevation data. Each grid point on the map has an associated elevation data value.

(3) Building data. This data describes, in 2D, the buildings outline viewed from above.

(4) Tile data. Tiles are used to represent roads, vegetation, water, etc. These are described as 2D outlines viewed from above.

The remaining data in the UCCATS terrain file is not used. The UCCATS file is a binary file created on a VAX using FORTRAN. Since the project is UNIX based, this data file is converted into a usable form, where the UCCATS data file is preprocessed to generate an ASCII data file to be used by the NUCAT project. Incorporated in the pre-processing step are:

a. Conversion of VAX Binary to UNIX Binary

The VAX stores bytes in what is known as "Little-Endian" order, where the most significant byte is in the low order part of the word. UNIX or IEEE stores bytes conversely, in "Big-Endian" order, where the most significant byte is in the high order part of the word. Therefore, every float, integer, and character read from the UCCATS file must be converted into the format used by UNIX.

If the VAX word is represented as such:

v0	v1	v2	v3

Then the corresponding UNIX integer and character would be:

v3	v2	v1	v0

And the corresponding UNIX float would be:

v1 -1	v0	v3	v2

b. Conversion to Meters

All data values specified in kilometers are converted to meters. This is done so that all data values are in consistent units allowing easier manipulation of the data.

c. Conversion of XY to XZ Data

The XY-based elevation data is converted to a XZ-based format. The elevation data is then easily represented in a 3D coordinate system rather than continually converting the 2D representation to 3D. All data in the file is represented in 2D. The elevation data is based on an XY grid with the origin at the lower left corner. In converting to a 3D version, an X-Z grid is used with the origin at the upper left. Because of the difference in location of the origin, the X indices remain unchanged, but the Y indices are interchanged to properly map their new Z indices. When interchanged, the Y_0 row becomes the Y_n row, the Y_1 row becomes Y_{n-1}, Y_2 becomes Y_{n-2}, and so on (Figure 2-4).

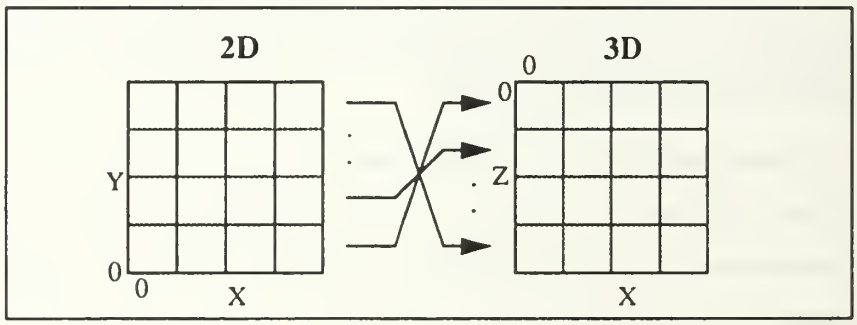

Figure 2-4. Conversion of 2D Elevation Data to 3D

d. Force Counter-Clockwise Vertex

All building vertices are written in counter-clockwise manner. The building vertices represent the outline of the building as viewed from above. In the UCCATS data file, some of the buildings are not represented with their vertices in counter-clockwise order. Therefore, each list of vertices is checked to see whether it is in counter-clockwise order. If so, then they are written to the output file in the same order.

If the vertices are not counter-clockwise, then they are written in reverse order. Writing the vertices in counter-clockwise order enables the use of backface removal when rendering the buildings.

The conversion process of the UCCATS data file is made more complex by the record headers which VAX FORTRAN places into the file. This creates difficulties when a continuation record is written. The following six record headers need to be scanned for and discarded: 1) start of file, 2) integer/char start record, 3) float start record, 4) integer/char continuation, 5) float continuation, and 6) end of file.

2. Radiosity Data

A radiosity data file is generated by a radiosity tool package specifically written to support NUCAT. The radiosity tool is discussed further in Chapter 4. Each file defines the radiosity data for one floor of a building. This binary file contains the data buffers used by the radiosity package in rendering the scene. These buffers include: radiosity and viewing parameters, the element points buffer, the elements buffer, the patches buffer, the colors used, and the vertex radiosity buffer.

To use this data file, the NUCAT project must initialize the radiosity parameters and read the data file into the radiosity buffers. To view the radiosity data, a procedure is called which renders all elements in the scene. Note that only one radiosity file may be loaded at a time. Use of multiple radiosity data files will require each file to be read in and loaded if it is not the current one in use.

3. Elevation Boxes

The elevation box data file is an ASCII file that describes the elevation boxes for the map file in use. Elevation boxes are defined to contain a set of contiguous elevation points not equal to the ground plane, where these points are then specified as belonging to some XZ bounded box. Thus hills, ridges, and gullies can each be defined by one elevation box. This file is created by a pre-processing step which works from the elevation data. Detail explanation of how this file is created is given later in Chapter 4. Creating these boxes decreases the amount of elevation data, co-planar to the ground plane, that is being tmeshed. Whereby a single plane can be used for the ground plane and only those elevation points not equal to the ground plane (elevation boxes) are tmeshed.

4. Interior Building Models

For the particular terrain data file being used (UCCATS), the only data provided for a building interior is the number of floors and the total number of rooms. There is no data for generating detailed floorplans or room contents. For the project, a sample test floorplan is used to test the interior building

concepts and algorithms. This sample floorplan is modified to fit into any four-sided building that the user enters. This process is detailed later in Chapter 4. However, a more detailed and complex floorplan is desirable. A format for representing the interior building data has been designed and will be used when a detailed floorplan becomes available. Currently, we are looking at some building files supplied by UC-Berkeley [Sequ90].

5. Texture Files

In the NUCAT project, textures are used to add realism to the image. The textures are mostly used to represent different building materials. The material of a building is visually determined by the texture. The different textures help distinguish buildings that are located close to each other but made of different materials. In addition to buildings, terrain elevation and road tiles are textured. All textures used are supplied by GIF files which are read in and defined during initialization.

III. NUCAT OVERVIEW

A. NUCAT ORGANIZATION

The NUCAT project is composed of several components linked together: navigation, visibility determination, radiosity, and rendering software. The navigation and radiosity parts are compiled into separate libraries which are linked with the NUCAT project. The code for managing and manipulating the model floor plan is also compiled as a separate library.

B. NUCAT PROGRAM FLOW

The step performed by NUCAT is initialization. During this step, all necessary data files are read and parsed, all object chains are created, and the map grid cells are populated. Then, in the main rendering loop the following steps are performed:

(1) in/out building determination and loading of interior building model

(2) determine which map cells are visible

(3) render the contents of all map cells which are visible

(4) if inside a building, render the building interior

The map cell containing the user will always be drawn. When the user is inside a building, that building's interior model is also drawn, which allows the outside world to be visible from inside the building through a window.

C. HIGHLIGHTS

The goal of NUCAT is to provide a method and basis for movement in and around an urban environment and for displaying its contents. The primary factors involved with this are construction and modelling of the urban environment, determining what is visible, performing the walkthrough of any building entered, and using the radiosity data for building interiors. These factors are discussed below.

1. Virtual World Model

One factor is modeling the structure and contents of the virtual world. The urban environment contains elevation data, buildings, vegetation, roads, etc. from the UCCATS data file. This file data is then modified into a generic representation for use with NUCAT, which uses a pre-processing step before

executing NUCAT. This step can be reproduced for other source data files of different formats, as long as the data file is converted into the format used by NUCAT.

The nature of the data is of two forms: exterior or interior. Exterior refers to things outside a building, where interior refers to inside floorplans and contents of each building. From the UCCATS data file, three common data types are grouped together: elevation data, buildings, and tiles, which are all linked to respective lists.

The virtual world is a map divided into a number of grid cells. Each grid cell has a list of which elevation boxes, buildings, or tiles that occupy any of its x-z space. Pointers to the contents of the grid cell are used when rendering that specific grid cell. Currently, the map is divided into a 50x50 grid.

The models used for representing the interior of buildings were investigated, but due to the lack of complex interior building data, a comprehensive evaluation cannot be accomplished. Currently, the building data is represented as a hierarchy of floors, room, and contents.

2. Visibility Determination

Maintaining the real-time aspect is important. This requires improved methods for determining what is visible. The basic techniques of backface removal and zbuffering are not sufficient enough to accommodate the great number of polygons rendered in a real-time manner. Therefore, a method is needed to cull large portions of the data set before the polygons are sent through the graphics pipeline.

For visibility, the terrain map is divided into grid squares, where each square is populated with objects. An object may be present in multiple grid cells if that object crosses over the boundaries of several grid cells (e.g. a long road tile). The contents of only those grid cells visible to the view volume are drawn. Two methods exist for determining which map cells are visible: 1) the ray-clip method and 2) the hemi-pixel method. The ray-clip method uses a combination of ray tracing and line clipping to create a box of visible map cells for the current view volume. The hemi-pixel method projects the current view volume onto a hemi-plane representing the map grid. Each pixel of the hemi-plane buffer is then scanned to determine which map cells are visible. These methods are discussed in detail in Chapter IV. Note that these methods are only for determining what is visible for the outside world, not for inside buildings. Figure 3-1 helps explain this concept. The figure represents a top-down view of the map, where the shaded area represents what the user can see. Therefore, only those map cells visible to the user are drawn.

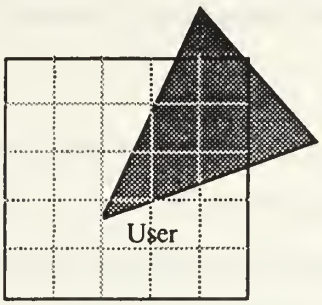

Figure 3-1. Visible Map Cells

Resolution is used for controlling the amount of detail shown. For each map cell drawn, a near/far resolution is determined in relation to the user's viewpoint. For near resolution, objects are textured and drawn in full detail; for far resolution, no texturing is done and less detail is shown for the buildings.

3. Walkthrough

The connection of the interior building model and the exterior map model are combined in a manner transparent to the user. This allows the user to move into a building and to look out of a window and see the outside world. Two separate models being used, one for the inside of the building, another for outside, each specific for its own use.

Whenever the user is outside, all buildings in the user's map cell are checked to see if the user has moved inside one of them. If the user is currently inside of a building, then only that building is checked to see if the user has moved outside. Once inside, whatever internal representation is available for that building is used, which can be a radiosity model, a polygonal model, or nothing at all.

The exterior map model of all visible map cells is always drawn. Then, if the user is inside a building, the building's interior model is drawn.

4. Radiosity

The radiosity data used by NUCAT is created as a pre-processing step, where NUCAT reads in this data and uses the radiosity renderer to display this data when inside the associated building. The radiosity program used for generating the data is based on Chen's (Arvo91), but is modified for functionality and use with this project.

The core parts of the radiosity tool from Chen's implementation are the data structures, scene description format, and basic routines for calculating and distributing the radiosity. However, Chen's model

allows neither simple scene modifications or interactive input to the process, which is the focus of the modifications. The following changes are made to the radiosity tool:

a. Scene data file

Modified first is the scene description code. Previously, the scene was defined as hardcoded global variables. Thus, for any changes to the scene the code needed to be recompiled. To avoid this problem, the scene description information is moved into a data file format which is usable as input to the radiosity tool. Thus, different scene data files are used without being recompiled.

b. Radiosity data file

This involves the creation of a data file that saves the current radiosity scene for use later on. Previously, the user started from scratch for each invocation. To solve this, current radiosity calculations are saved to a file rather than being lost when exiting the tool. This data file can be used later either to display the current radiosity scene or to continue distributing radiosity. This data file is the radiosity data file used by the NUCAT project.

c. Interactive Input

Pop-up menus are added for the user to interact with and control the tool rather than merely viewing the results of the radiosity calculation. The user can then move around the scene and view it from any position while radiosity calculations are being made. The user can also control the process of radiosity calculation by beginning, suspending, stepping one frame at time, or resetting the entire process. The user also controls over various display options such as flat or Gouraud shading, whether to use the ambient value, or showing the shooting patch. The user can then save the current radiosity data or read in such a data file for processing.

d. Gouraud Shading

The Chen model provides only flat shading. A new data structure and code were added to allow for the calculating and rendering of vertex radiosity values using Gouraud shading, which provides a scene more realistic than the flat shaded model. The Gouraud shading smooths the aliasing effects caused by the hemi-cube.

IV. NUCAT DETAILS

A. OBJECTS

1. Elevation Boxes

a. Motivation

To tmesh an entire map of elevation data is a time consuming process, where a map may contain a 400x400 grid of elevation points. To solve this, all elevation data not equal to the ground plane is grouped into separate boxes, elevation boxes. Then, a single polygon is used to represent the ground plane, thus the elevation boxes alone are tmeshed. The use of elevation boxes combined with visibility determination greatly increases the speed of rendering the elevation data. Figure 4-1 shows the areas of a map which would be contained by the elevation boxes.

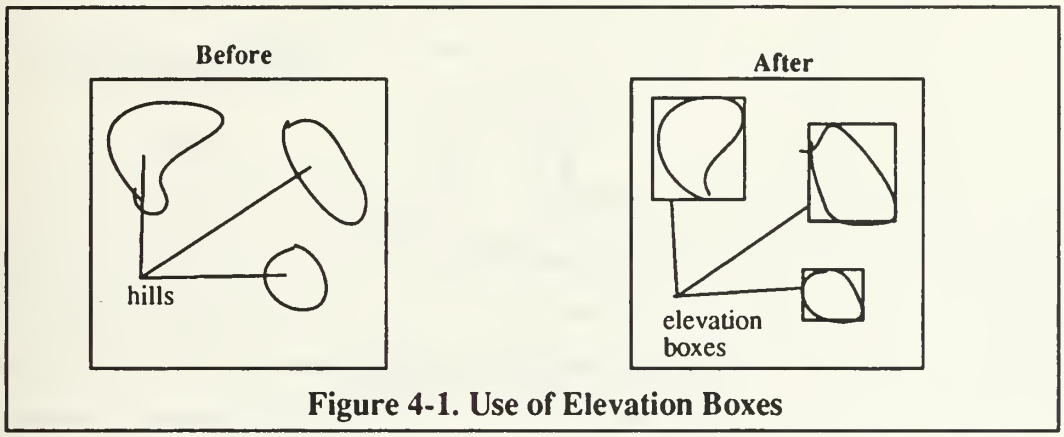

Figure 4-1. Use of Elevation Boxes

b. Creation

An elevation box outlines a set of elevation points which represent objects such as a hillside or gully. These box boundaries are created as output from a pre-processing step of the elevation data found in the UCCATS terrain data file being used. Once the elevation data is read into a two-dimensional array, the array is then processed in column order to produce the boxes. Each elevation point is checked until one is found which does not match the given ground plane, and each point next to this one is then checked as well. The order in which these four points are checked is first left, up, right, then down. The box is expanded to include each point which does not equal the ground plane, where the process is recursively followed for each point found to belong to the current object such as a hill. The process ends when all points belonging to the

object have been checked and no new points are found. This box is then written to the output file. After this, the scanning process begins again at the first point of the object. Whenever an elevation point is checked, it is marked. Thereby, only non-marked points are processed and/or added to the object.

c. Rendering

Rendering of an elevation box is a more complex than it may seem. First, all elevation points in the box are drawn using a tmesh, although this may cause some z-buffer tear with the ground plane, since the box encloses only the boundaries of the hill, where many points may remain at ground level (Figure 4-1). To avoid z-buffer tearing with the ground plane, checks are done to ensure that no co-planar tmesh triangles are drawn to occur with the ground plane. Also, some checks are added to turn tmeshing on/off should the hill not be entirely concave in nature (i.e. a "C" shaped hill). A "C" shaped hill with tmesh continually on would erroneously cause polygons to be drawn between the legs of the C. This process is depicted in Figure 4-2.

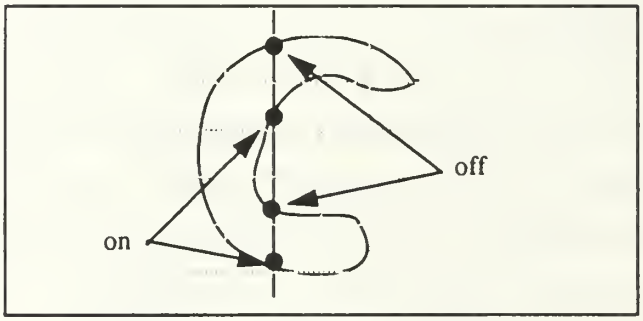

Figure 4-2. Tmesh Switching

The box is drawn X by Z, from left to right, bottom to top. Figure 4-3 depicts the tmesh order for any particular column X. For every loop iteration, six points must be tested, even though only two at most may be drawn. If the current loop points (3 & 4) belong to a triangle which has any points not equal to the ground plane, then those points are drawn. Thus, only those tmesh triangles that have all three points equal to the ground plane are not drawn.

The tests are as follows:

(1) If points 2, 3, 4, or 5 != to ground plane Then points 3 and 4 are drawn. (triangles b,c).

(2) If point 1 is != to ground plane Then draw point 3. (triangle a)

(3) If point 6 is != to ground plane Then draw point 4. (triangle d)

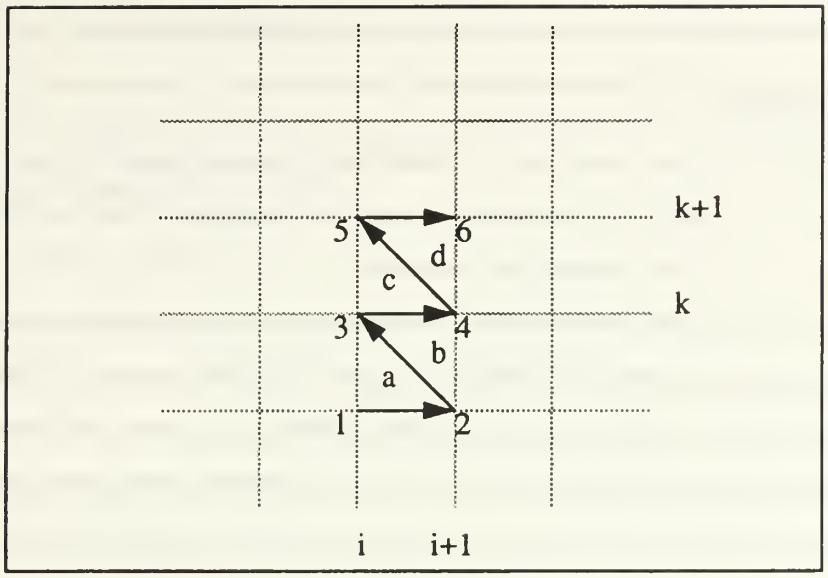

Figure 4-3. Elevation Grid

2. Buildings and Tiles

a. Creation

Since the buildings and tiles in NUCAT are created similarly, they will be discussed together. In the UCCATS terrain file, buildings are represented as a list of X-Y points, where each point is relative to the map origin (lower left corner). These points, along with the building height and elevation data, are used to create the 3D buildings for NUCAT. A polygon is created for each two consecutive points in the list, where the polygon and the building are the same height. The elevation origin of the building is determined by calculating the elevation of each point for the building. Then the lowest such value is used as the base elevation for the building in order to sink the building to its lowest point overall. For example, if the building is located on a hill, it appears to sit inside the hill a rather than on top. The roof of the building is merely the 2D points raised to the proper elevation.

b. Rendering

When rendering buildings, each polygon in its list is drawn first and followed by the roof. Backface removal is used to avoid drawing the unseen polygons on the other side of the building. The material and texture appropriate for that building type is set when drawing the polygons. Before any of the building is drawn, its shadow is first drawn (Section 4.D.2). These methods allow the outside of a building to be viewed.

When inside a building, the radiosity or polygonal model is drawn if the building is four sided. If the building does not have four sides, then just the exterior polygons for the building are drawn.

B. NAVIGATION

Navigation through the virtual world is bundled into a separate compile unit and linked in with the overall project. This separate package controls movement and orientation of the view volume. The NUCAT project controls the user input and feeds it to the navigator.

The user is able to control the direction of movement, the view orientation, and the rate of speed for movement. Input devices used for controlling these aspects are defined in detail in Appendix A.

The user navigates through the virtual world according to the eye-in-hand model. The navigator performs the following functions: setting the current movement direction vector, rotating the view about XYZ axis, moving and setting the view volume based on current speed vector and CPU clock time, drawing an orientation cross-hairs, and either initializing or resetting any navigation parameters.

To enable use of the navigator, the following navigation parameters are initialized: the initial view point, the speed vector, and the mouse if used for navigating. The navigator then sets the view volume based on the current rotation, direction, and speed as supplied by NUCAT.

Continually, frame by frame, the navigator updates the view volume. This involves loading the current rotation matrix, then multiplying by the relative direction vector to get a world coordinate direction vector. The view point and reference point are then updated by this direction vector multiplied by both the speed and the delta cpu time from the last update. Finally, a correction is done for the roll value if the "up" vector is upside down; this is required since the IRIS GL does not model any up vector for the view volume. If a roll correction is not done, then the screen image wrongly flips over when the user turns upside down. Figure 4-4 shows the code sequence for these operations.

```
loadmatrix (rotation);
multmatrix (direction vector);
getmatrix (direction vector);
viewpoint = viewpoint + (delta time * speed * direction vector);
refpoint = refpoint + (delta time * speed * direction vector);
set roll for current up vector position
```

Figure 4-4. View Volume Update

The direction vector is relative to the user's current orientation, so that the user may be looking in one direction while travelling in another.

When navigating via the mouse, the cursor position is polled and compared to the window edges. When the cursor is near an edge, the appropriate rotation is done; the closer to an edge, the larger the rotation. Forward and backward movement is accomplished by using the left and middle mouse buttons.

The navigator is constrained by the CPU clock and not from the frame rate. Thus, consistent speed is maintained though the frame rate may vary widely.

C. VISIBILITY DETERMINATION

The process of visibility determination determines which map cells are visible based on the current view volume. Two methods to address this are: the Ray-Clip method and the Hemi-Pixel method, both of which are described below. Using these methods, NUCAT draws the contents of all the map cells currently visible. Note that the number of grid cells may affect the speed of either method (i.e. 15x15 grid map v.s. a 50x50 grid map); thus, the greater number of grid cells increases the time required for each method to calculate the visible map cells. See Section 4.C.3 for an in-depth comparison of both methods.

1. Ray-Clip Method

The ray-clip method is used for exterior visibility determination. Ray-tracing[Glas89] and line-clipping[Fole90] are used to determine which map cells are visible based on the current volume. The method produces a XZ bounding box that contains all visible map cells. This box is created in two steps: a ray intersections step and then a box clipping step. These two steps are described as follows.

a. Step 1 - Ray Intersection

Four unit vectors are defined based on the view-volume definition (fov-y, aspect ratio), where the view volume is located at the origin and looks down the -Z axis. These vectors are calculated once, and they are then oriented to correspond to the current view volume. This is accomplished by multiplying the vectors by the current rotation matrix for the view volume. Next, each ray is projected and an intersection point with the ground plane is generated. The following equation is used to calculate each intersection point:

$$t = \frac{-(P_n \bullet R_0 + D)}{P_n \bullet R_d}$$

Where $\mathbf{P_n}$ is the ground plane normal, $\mathbf{R_0}$ is the ray origin, $\mathbf{R_d}$ is the ray direction, and D is the distance from the origin to the ground plane, $\mathbf{R_d}$ is unique for each of the four rays and $\mathbf{R_0}$ is the view point location. This step generates zero to four intersection points dependent on the view volume orientation (Figure 4-5). Three simple cases are shown as a general case. The Vs represent a side view of the view

volume. Case 1 will generate two intersection points, case 2 will generate no intersection points, and case 3 will generate four intersection points. One or three intersection points may result from the view volume is being rolled.

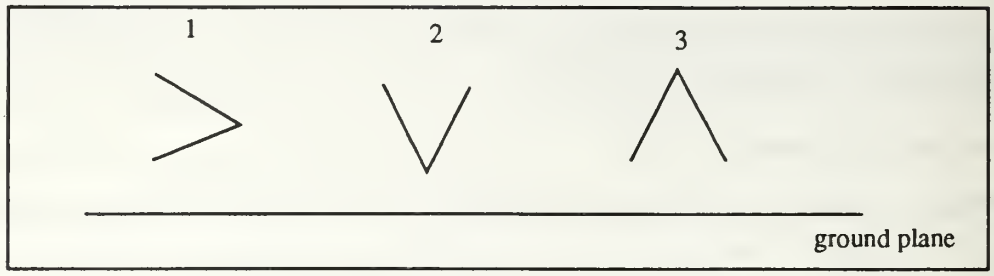

Figure 4-5. Ray Intersections

If the view-volume is facing up (case 2), then no intersection points are generated and no view box is returned. If the view-volume is facing down (case 3), then most likely four intersection points are generated and a fairly precise view box is generated. The precision depends on the skew about the Y axis or the view volume in relation to the map grid. Most likely, the view-volume is oriented level to the ground plane (case 1), so that two intersection points are generated. In this case, the user direction is used to determine which side of the map boundary is used for the other two points for the box. The direction vector is the view point to reference the point vector. This can cause a large over-estimation of the maps cells visible.

b. Step 2 - Box Clipping

After the intersection points are calculated, they are grouped with the view point to form an XY box that contains all of the points. This box is then clipped to the size of the map area being modeled. This clipping is based on the Cohen-Sutherland line-clipping algorithm [Fole90], where the lines being clipped are the XZ box boundaries and the window being clipped to is the physical map boundaries. The final clipped box returned contains all visible map cells.

Formation of the XZ box is dependent upon the number of intersection points available. A special case exists if only one or two intersection points exist (Figure 4-6). This occurs when the two points (labelled 1 and 2) are inside the map boundaries, and if this special case is not handled, then nothing beyond the intersection point to the map boundary would be seen. This is represented by the dark shaded region in Figure 4-6. To handle this special case, the view direction is tested and the intersection points are extended to the map boundaries appropriately (labelled 3 and 4) so that the lighter shaded region is rendered along with the dark shaded region. When an intersection point is extended, a vector is created pointing from the view

point to the intersection point. If this vector points in the +X direction, then the max-x map value is used for x; otherwise, the min-x value is used. The same is done for the Z value. If the vector points in the +Z axis, then the max-z value is used for Z and the min-z value is used if it points in the -Z direction. In the Figure 4-6, this method generates a box with opposite corners at v and 3.

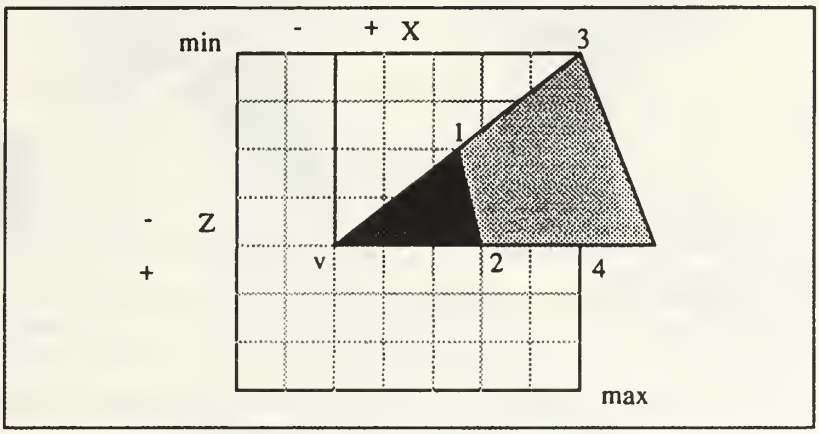

Figure 4-6. Case 1: Two Intersection Points

2. Hemi-Pixel Method

The hemi-pixel method provides a more precise representation of the visible map cells, and since less speed map cells are rendered, the frame rate increases. This method borrows a technique from radiosity know as the hemi-cube, although a single hemi-plane is used rather than the whole hemi-cube. The view volume is projected onto the hemi-buffer and then the pixels are scanned to see which cells are visible. For Figure 4-6, rather than generating a box with several unneeded non-visible map cells, this method allow only those map cells in the triangle 3v4 to be rendered.

For this method, a window is created to represent the hemi-buffer. Its pixel size must match with the map grid size. For example, if the grid is 50x50, then the pixel size of the hemi-buffer is also 50x50. This is necessary as each pixel of the hemi-buffer represents one map cell.

To determine which map cells are visible, the hemi-window is set and cleared to some background color. A top-down orthographic projection of the current view volume is then projected onto the hemi-plane. A color different from the background color is used for the view volume. The hemi-window (its framebuffer) is then read into a buffer which is then scanned one pixel at a time. If a pixel is not equal to the background color, then the map cell for that pixel is visible and its contents are rendered.

These steps are shown in Figure 4-7. Step 1 shows the 3D orientation of the view volume over the map grid. Step 2 shows the view volume projection onto the hemi-plane. Each pixel of the hemi-plane is checked to see if the view volume covers some part of it (i.e. pixel is not the background color). Step 3 shows the result of the scan and which map cells are visible.

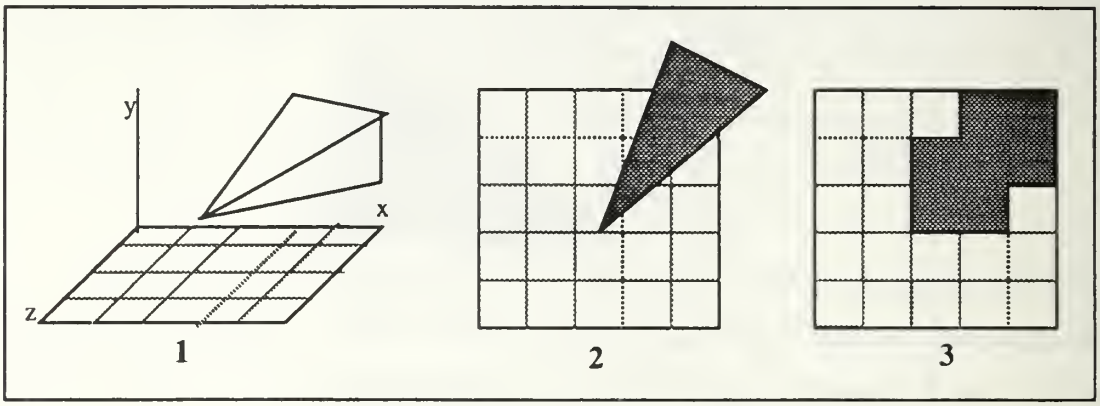

Figure 4-7. Hemi-Pixel Method

3. Comparison of Methods

Both the ray-clip and hemi-pixel methods operate at nearly similar speed when viewing the entire map. The hemi-pixel gains the edge when viewing portions of the map not aligned on the XZ axis. This is due to the fact that the ray-clip method is including extra map cells that are not visible (Figure 4-8).

The number of grid cells the map is divided into may affect the performance of both methods. Using the ray-clip method, more map cells are drawn than necessary. This occurs because an XZ bounding box is being used to enclose all visible map cells, which may include extra map cells not visible (Figure 4-8). The shaded area in the figure is what is visible. These are the only cells drawn by the hemi-pixel method, whereas the ray-clip method includes the entire map grid. The geometry of the ray tests do not change, so the time for generating the view box remains constant; yet, if more grid cells are used then there exist extra map cells which are included in the visible box. This is time consuming since map cells not visible are drawn. Fewer grid cells generate fewer unneeded map cells in the visible box allowing for a more broad resolution.

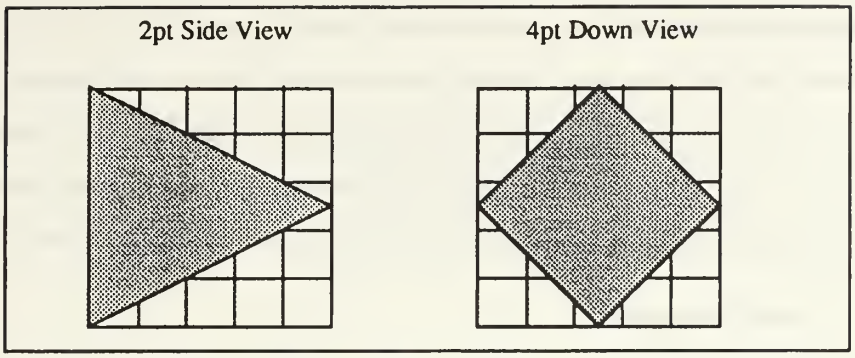

Figure 4-8. Extra Map Cells

For the hemi-pixel method, dividing the map into more grid cells affects the pixel operation of the method (reading and scanning the frame buffer). The hemi-pixel method, however, avoids the problem of extra map cells being included, since only those cells covered by the view-volume projection are drawn, although the method requires every pixel to be scanned and checked.

Fewer grid cells allow for a larger map area to be represented by each grid cell. Objects may then be present in fewer map cells -- perhaps in one. Thus, for each map cell there are more objects to be drawn, some of which may not be visible. If more grid cells are used, then objects are spread over more map cells where each cell is less populated. The greater number of map cells allows for finer resolution of the viewing area but also requires more checks to be done.

In the end, a balance must be found between the cell population and the number of grid cells, where more map cells provide better resolution for both methods.

D. IMAGE REALISM

1. Textures

Textures are used to add visual reality to the scene and to help distinguish buildings of different materials. To date, all buildings, terrain, and tiles, except vegetation, are textured. All textures are read in and defined as part of the initialization phase. When texturing objects, the frame rate slows down about 1 frame per second (based on a 4D/340 VGX IRIS). A resolution factor is used to determine when to texture various objects. A texture becomes less distinct the further away it is from the viewer; thus, if the distance from the user to the object is greater than the resolution factor, then the object is not textured. The resolution factor must be far enough from the user to allow the addition of a texture, to a previously untextured object, to go unnoticed by the user (i.e. it does not suddenly pop up on the surface). Also, it must be close enough so that

some increase in frame rate occurs by not texturing all the objects. In addition, backface is used so that only those polygons visible are textured.

When using four-component texturing, the modulation option is used to maintain proper lighting of the polygons. Decaling is a quicker option, but causes no lighting effects to be seen for the decaled polygon. Unfortunately, when using modulation, the polygon color darkens more than desired. Therefore, in order to maintain the original polygon color when texturing, the polygon color must be artificially lightened.

a. Object Texturing

Buildings - all exterior perimeter polygons are textured. The length of the polygon determines the repetition factor of the texture. Note that all such polygons are quadrilaterals. Each building material has a different texture.

Tiles - To date, only road tiles are being textured. For each vertex of the tile, one of the texture corners is used. Note: there may be more than four points for a tile. The enlongated nature of the tile allows the texture to be stretched across the polygon, creating a nice streaked effect.

Terrain elevation - while tmeshing the elevation points, the texture is spread out to cover a 2x2 grid patch of elevation. The ground plane is textured with a fixed repetition over the entire map. However, a mapping must be done from this overall map to the actual portion that is visible, so that the visible part is drawn and textured rather than the entire map.

2. Shadows

To add more realism to the buildings and to help the user discern where the sun is located, simple shadows are drawn for each building. These shadows are created by projecting the building's outline from the sun angle onto the ground plane. The shadows are drawn with zbuffer and backface on and before the actual building is drawn. No geometric tests or corrections are performed where the shadow falls. Therefore, all shadows are flat and coplanar to the ground plane. Because of this, some shadows may appear to slide under nearby hills or buildings upon which they fall. A black half-tone pattern is used for filling the shadow polygons.

For generating the shadow, the shadow pattern and color are set and a shade matrix is calculated, which converts the building coordinates into their appropriate shadow coordinates. The current matrix stack is pushed and the shade matrix multiplied to it. Each polygon that represents the exterior perimeter of the building, and the roof is then drawn. Finally, the matrix stack is popped and the default pattern reset.

The shade matrix is based on the y-plane to be projected on and the light position (lx,ly,lz) to be used. It is composed of a scale part(sx,sz) and a translation part (tx,ty,tz) and is calculated as follows:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ sx & 0 & sz & 0 \\ 0 & 0 & 1 & 0 \\ tx & ty & tz & 1 \end{bmatrix}$$

where

sx = - light_x / light_y

sz = -light_z / light_y

tx = (yplane*light_x) / light_y;

ty = yplane;

tz = (yplane*light_z) / light_y;

Warning: on the IRIS Indigos, none of the identity cells can be zero. Since the Indigo attempts to create an inverse matrix whenever the matrix stack is multiplied, an error for the zero identity cell occurs. This is avoided by using a very small number rather than zero (like .00001). This avoids the error while maintaining the near correct results.

This process is geometrically represented in Figure 4-9. In the figure, point A represent the shadow point of point P as determined by the scalar portion of the above shadow matrix. The translation part of the shadow matrix caused point A to be translated to point B.

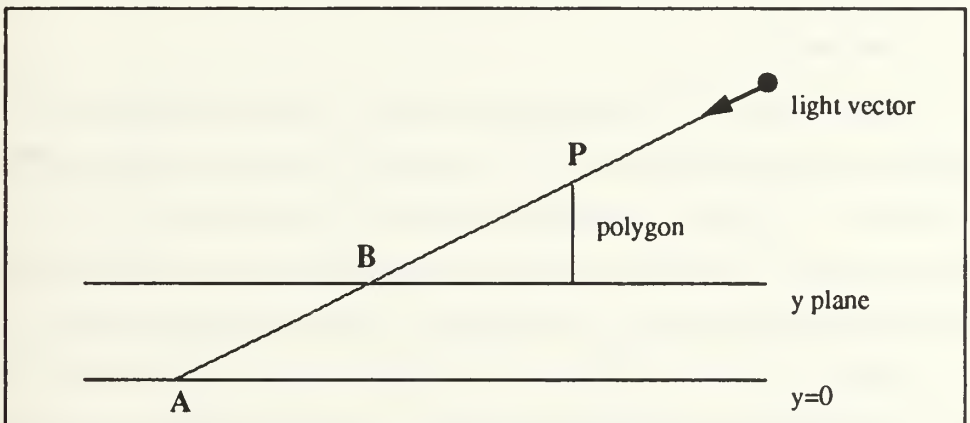

Figure 4-9. Shadow Generation

E. WALKTHROUGH

For the building walkthrough, three issues need to be solved: representing the interior building model, determining the radiosity lighting of the interior, and determining what is visible inside the building. The first two issues were accomplished, but the third was not due to lack of a complex building interior to adequately test the algorithms.

1. Radiosity

The radiosity data files generated by the radiosity tool are used to represent the interior lighting of the building. To use these data files, a link is created between the radiosity tool and NUCAT which enables NUCAT to access the radiosity data buffers and rendering procedure of the radiosity tool. This first involves bundling the radiosity tool into separate compile units which are compiled and placed into a library. This library is then linked with the NUCAT project. Simplicity was maintained in the process, so that any program can easily make use of the radiosity library.

All that NUCAT needs to do is to include the external radiosity header file and provide the radiosity data file to be used. NUCAT first calls the proper routine to read in and load the given radiosity data file. Then, inside the display loop for NUCAT, the routine for displaying the radiosity data is called. Thereby, the radiosity data becomes another 'object' to be drawn with all others in the display loop.

This setup works exceeding well. The radiosity data is only used when the user is inside a building. Display of the radiosity data only causes about a 1 frame per second slow down (4D/340 VGX). Also, a variety of radiosity data files, representing different floorplans, can be used easily together or swapped in/out.

2. Model Floorplan

A model floorplan was created to test out the various walkthrough algorithms. Also, a radiosity file was generated from the floorplan for use in the program. The floorplan consists of a hallway with two rooms off each side. This floorplan is used for any four-sided building that the user enters. Only four sided buildings can be used, since this allows the floorplan to be transformed into the building being entered. For example, when entering a long building, the floorplan will be stretched to fit the building's dimensions.

Whenever a four-sided building is entered by the user, a conversion matrix is calculated to transform the model floorplan to fit the building entered. The conversion matrix is composed of three basic parts: translation, rotation, and scale. The translation part translates the model floorplan from the origin to the

building's map location. The rotation part rotates the floorplan to the same orientation as the building. Finally, the scale part scales the floorplan so that it fits exactly the building's dimensions.

Therefore, when the building's interior is drawn, this conversion matrix is multiplied onto the current matrix stack. The floorplan's radiosity data file or polygonal file is then drawn to represent the interior.

V. NUCAT SUMMARY

A. CONCLUSION

The NUCAT project defines a mechanism for constructing and rendering walkthroughs of urban terrain. A flexible framework exists for linking the topics of radiosity, walkthrough, and virtual worlds, all of which is done in real-time on commecially available graphics workstations.

Methods for maintaining the real-time aspect of the project are generated for many areas of the project. This includes the implementation of two visibility algorithms and the use of near/far resolution for controlling the amount of detail given.

During the course of this project, much work was done in creating support tools to help further the investigative process. The following tools were generated: a fully interactive radiosity package; a BSP tree package; a generic navigation package; and a generic polygon management tool for creating and editing the model floor plane.

One of the main limitations throughout the project was the lack of a complex building interior model for use. Creating such a model from scratch would be a time-consuming task. Thus, only a small sample model floorplan was generated for use. To date, building models from UC-Berkeley are being investigated for use in the project. However, parsing and rendering software needs to be generated to process these file formats.

B. FUTURE WORK

1. Terrain Data Files

Additional data files of different format and content may be added for use. This would require new code for a pre-processor to parse the new data file and generate a NUCAT specific data file. Also, testing the NUCAT project using other different UCCATS data files would be helpful, especially a file which covers a larger map area and has rich content of objects. This would test the limits of the visibility routines used in the project.

2. Radiosity

Add the capability to load and use multiple radiosity data files. This can be done by making each building store its own radiosity data buffers.

A method for calculating radiosity on a room by room basis could be created. This would have to consider the latent radiosity for portals and project this energy into a room when the door is open.

3. NPSNET

The work accomplished by NUCAT can be linked in with NPSNET. This would probably involve incorporating the walkthrough and radiosity parts of NUCAT for use in the buildings used by NPSNET. The NPSNET buildings may have to be converted to the building format used by NUCAT. Also, the buildings may have to be populated with contents (i.e. furniture, etc.) if none exist. Finally, the radiosity tool can be used to generate radiosity data files for each of the NPSNET buildings used.

4. High-Resolution Combat Models

The combat models used by UCCATS could be linked in and used with NUCAT. This project focused on the 3D visualization of the terrain data used by UCCATS and did not use any of the combat models. Now the NUCAT project could be linked in with the UCCATS combat models to provide a 3D urban combat trainer.

C. PERFORMANCE

The polygons used in NUCAT are divided up into two sets: exterior and interior. The exterior set consists of 655 polygons and 13,154 tmesh triangles. Not all of the polygons for this set are convex, about 270 of them being complex. The interior set is the set of polygons used to represent the interior building model. This set consists of 336 polygons for the building model and 1752 polygons (elements) for the radiosity model. Table 2 below shows the frames per second for a variety of situations based on a 4D/340 VGX IRIS.

Table 2: PERFORMANCE TIMINGS

	Texturing	Radiosity	# Polygons	fps
View entire world	off	n/a	13809	5.03
View entire world	on	n/a	13809	2.75
Inside building	off	off	14145	6.67
Inside building	off	on	15561	3.33
Inside building	on	off	14145	2.51
Inside building	on	on	15561	1.35

The maximum number of polygons drawn for any case occurs when viewing the entire world. Therefor, the frames per second for this case represent the slowest times possible, since when moving through the virtual world, a large portion (50%+) of the data set will be culled when determining the visible data set to be rendered. When inside a building, additional polygons are added to the data set to be culled and rendered. The number of polygons added is dependant on whether radiosity is being used when inside the building. When radiosity is used, the number of elements in the radiosity scene is added to the data set. When radiosity is not used, then the number of polygons for the building model is added.

APPENDIX A: NUCAT User Guide

A. COMMAND LINE ARGUMENTS

The nucat command line is as follows:

```
nucat [-d -h -s -t -v]
```

The optional command line arguments must be delimited by at least one space and are detailed below.

`-d` Debug. Print out debug information.

`-h` Help. List command line options.

`-s` Small. Make the viewing window half screen size.

`-t` Texture. Read in and use texture files.

`-v` Version. Show version of nucat.

B. INPUT DEVICES

1. Mouse

The mouse is mainly used for selecting options from the pop-up menu. These options are detailed in Section C. The mouse is also used for navigation. By default, the mouse is not activated for navigation and the user must select this via the menu (see Figure A-3).

When used for navigation, the mouse cursor controls pitch and yaw orientation. This occurs when the cursor is moved towards the edge of the screen; the closer to the edge, the more pronounced the rotation. Also, the LEFT and MIDDLE mouse buttons can be used to move forward or backward along the line of sight, where movement occurs when the buttons are held down.

Note that the RIGHT button still is used for the pop-up menu. Also, the mouse is compatible with navigation input from other input devices.

2. Keyboard

The keyboard has two basic functions:

(1) Navigation (via the numeric keypad)

(2) Setting of program flags

Note that several of the flags set via keyboard are purely of experimental nature and may not be part of the final product but are included here for completeness.

a. Navigation

The numeric keypad is used to set the direction of motion (but not orientation). Striking a key causes one step of movement in the desired direction. The space bar is used to begin continuous movement in the set direction and such movement is stopped by key 5. The distance moved in one step can be increased or decreased by the + and - keys (see Figure A-1).

2 - backward

3 - down

4 - left

5 - stop

6 - right

7 - down

8 - forward

9 - up

space - move continuously

+ - increase step size by 50%

- - decrease step size by 50%

b. Flags

Esc - exit the program.

* - toggle the high/low orientation of the 2D window

b - toggle display of the bird cross hairs

d - decal all textures

f - use flat shading for radiosity data

g - use gouraud shading for radiosity data

l - level off orientation, resets pitch to 0 (not fully implemented)

m - modulate all textures

3. Dial Box

The dial box is mainly used to control orientation of the user (i.e. pitch, yaw, roll). This is the only way of setting these values unless the mouse is being used instead (see Figure A-2).

Dial 0 - roll, Z rotation

Dial 1 - pitch, X rotation

Dial 2 - yaw, Y rotation

Dial 5 - fog density

C. MENU ITEMS

The are four basic sub-menus available off of the main menu. The main menu also has the basic reset and exit options.

1. Program Options Menu

Inc Step - increment step size by 50%; same as the '+' key.

Dec Step - decrement step size by 50%; same as the '-' key.

Fog - toggles use of fog

Empty - used to test newly implemented features; currently not used.

Light On - toggles use of light source.

Use Radiosity - use radiosity lighting when inside buildings.

Map Cell 2 - activates use of the view hemi buffer for determining map cell visibility.

Build Textures - toggles use of textures if available.

Mouse Move - toggles the use of the mouse for navigation.

Gouraud On - toggles the shade model between FLAT and GOURAUD.

2. Render Menu

Elevation -display the elevation data

Buildings - display of the buildings

Tiles - display of tiles (roads, vegetation, water)

Labels - display of id labels for the displayed objects.

Map Grid - display the map cell grid over top the map.

Portals - display the exterior building portals (windows, doors) Not fully implemented.

3. View Menu

Front - view map from the front

Right - view map from the right side

Back - view map from the back side

Left - view map from the left side.

Top - view map from the top.

4. Window Menu

Status - display the status window at top of the screen.

2-D View - display the 2D top down view of map.

Building Key - display key for building colors or textures.

Tile Key - display key for tile colors.

Terrain Key - display key for elevation colors.

D. REFERENCE TABLES

A summary of the above information in tabular form.

		* High/ Low 2D	- Dec Step
7 Down	8 Forward	9 Up	+ Inc Step
4 Left	5 Stop	6 Right	
1	2 Backward	3 Down	

Figure A-1. Keypad Layout

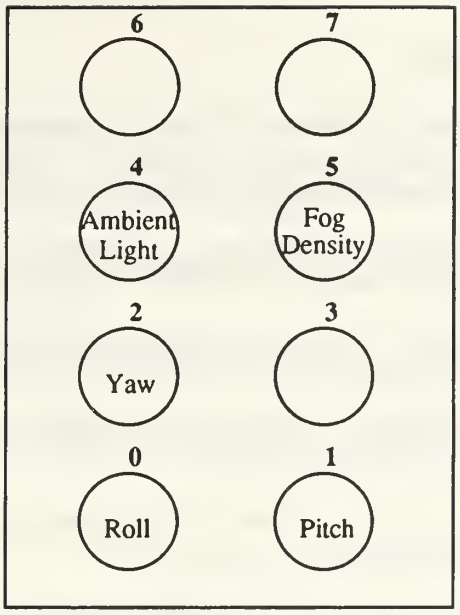

Figure A-2. Dial Layout

Left	Middle	Right
Forward	Backward	Pop-up Menu

Figure A-3. Mouse Buttons

LIST OF REFERENCES

A. VIRTUAL WORLDS

[Aire90] Airey J., Rohlf J., Brooks F.; "Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments"; ACM SIGGRAPH Special Issue on 1990 Symposium on Interactive 3D Graphics, 24, 2 (1990), 41-50.

[Broo86] Frederick Brooks; "Walkthrough - A Dynamic Graphics System for Simulating Virtual Buildings"; Proceedings of the 1986 Workshop on Interactive 3D Graphics.

[Clar76] James Clark; "Hierarchical Geometric Models for Visible Surface Algorithms"; Communications of the ACM; Vol 19, No 10; Oct 1976.

[Fuch83] Fuchs,Abram,Grant; "Near Real-Time Shaded Display of Rigid Objects"; Computer Graphics; Vol 17 No 3; Jul 1983.

[Funk92] Funkhouser, Sequin C., Teller S.; "Management of Large Amounts of Data in Interactive Building Walkthroughs"; SIGGRAPH'92.

[Garv88] Garvey R., Monday P.; "SIMNET (SIMulator NETworking)"; BBN Systems and Technologies; Bellevue, WA; Jul 1988.

[Rubi80] Rubin,Whitted; "A 3-Dimensional Representation for Fast Rendering of Complex Scenes"; ACM 1980.

[Tell91] Teller S., Sequin C.; "Visibility Preprocessing For Interactive Walkthroughs"; Computer Graphics (Proc SIGGRAPH'91); Vol 25 No 4; July 1991.

[Weil80] Weiler; "Polygon Comparison using a Graph Representation"; ACM 1980.

[Zyda92] Zyda M., Pratt D., Monahan J., Wilson K.; "NPSNET: Constructing a 3D Virtual World"; Proceedings of the 1992 Symposium on Interactive 3D Graphics; April 1992.

[Zyda91] Zyda M.; Graphics Notes - Book 10; 1991.

B. RADIOSITY

[Baum91] Baum D., Mann S., Smith K., Winget J.; "Making Radiosity Usable: Automatic Preprocessing and Meshing Techniques for the Generation of Accurate Radiosity Solutions"; SIGRAPH'91; Vol 25 No 4; July 1991.

[Baum90] Baum D., Winget J.; "Real Time Radiosity Through Parallel Processing and Hardware Acceleration"; SIGGRAPH; 1990.

[Chen90] S.E.Chen; "Incremental Radiosity: An Extension of Progressive Radiosity to an Interactive Image Synthesis System"; SIGGRAPH, Vol 24 No 4; Aug 1990.

[Cohe88] Cohen M., Chen S., Wallace J., Greenberg D.; "A Progressive Refinement Approach to Fast Radiosity Image Generation"; Computer Graphics Vol.22 No.4; Aug 1988.

[Puec90] Puech C., Sillion F., Vedel C.; "Improving Interaction with Radiosity-based Lighting Simulation Programs"; SIGGRAPH, 1990.

[Reck90] Recker R., George D., Greenberg D.; "Acceleration Techniques for Progressive Refinement Radiosity"; SIGGRAPH; 1990.

C. GENERAL

[Arvo91] Arvo, James; "Graphic Gems II"; Academic Press; 1991.

[Dobb90] Dobbs, L.; "The UCCATS Terrain Editor User's Manual, Ver 1.0"; Lawrence Livermore National Laboratory; Livermore, CA; 1 May 1990.

[Fole90] Foley, van Dam, Fener, Anger; "Computer Graphics - Principles and Practice"; Addison-Wesley Publishing Co.; 1990.

[Glas89] Glassner, A.S.; "An Introduction to Ray Tracing"; Academic Press; 1989.

[Sequ90] Sequin C., Smith K.; "Introduction to the Berkeley UNIGRAFIX Tools, Ver 3.0"; Report No. UCB/CSD90/606; University of California - Berkeley; November 1990.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
Dudley Knox Library Code 52 Naval Postgraduate School Monterey, CA 93943	2
Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943	2
Michael J. Zyda, Code CS/Zk Naval Postgraduate School Department of Computer-Science Monterey, CA 93943-5100	2
David R. Pratt, Code CS/Pr Naval Postgraduate School Department of Computer-Science Monterey, CA 93943-5100	2
Mark R. Boone Naval Surface Warfare Center Dahlgren Division K53 Dahlgren, VA 22448	4